WORD SEARCH USING KMP PARALLEL

PROGRAMMING PYTHON

PROJECT REPORT

For
Parallel Distributed Computing(CSE 4001)

Submitted by:

NAME	REG.NO
ROHAN BEJOY	20BCE2586
ZAMAN SALEEL	20BCE2025
KARTIK TRIPATHI	20BCE2098

Under the Guidance of

Dr. Hiteshwar Azad

WINTER SEMESTER 2020-21

1. ABSTRACT

While our interaction with computers and other technologyhas evolved in the recent years, the main form of communication with a computer still at heart remains by text. Data is still stored in the text format and is being processed by software in this format. Traditionally, word searching is done using serial computing and the performance using a single core is quite limited, hence we would like to implement one of the popular algorithms usedof string matching and parallelize it to improve the performance. In computing, approximate string matching (often conversationally stated as fuzzy string searching) is that the technique of finding strings that match a pattern (rather than exactly). the matter of approximate string matching is often divided into 2 subproblems: finding approximate substring matches within a given string and finding wordbook strings that match the pattern.

Traditionally, word searching is done using serial computing and the performance using a single core is quite limited, hence we would like to implement one of the popular algorithms used of string matching and parallelize it to improve the performance. By the number of patterns used, the algorithms are further divided. String matching strategiesor algorithms provide key role in various real-world problemsor applications. A few of its imperative applications are Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics, Digital Forensics and Information Retrieval Systems etc.

2. INTRODUCTION

While our interaction with computers and other technology has evolved in the recent years, the main form of communication with a computer still at heart remains by text. Data is still stored in the text format and is being processed by software in this format. Traditionally, word searching is done using serial computing and the performance using a single core is quite limited, hence we would like to implement one of the popular algorithms used of string matching and parallelize it to improve the performance. In computing, approximate string matching (often conversationally stated as fuzzy string searching) is that the technique of finding strings that match a pattern (rather than exactly). the matter of approximate string matching is often divided into 2 subproblems: finding approximate substring matches within a given string and finding wordbook strings that match the pattern. Traditionally, word searching is done using serial computing and the performance using a single core is quite limited, hence we would like to implement one of the popular algorithms used of string matching and parallelize it to improve the performance. By the number of patterns used, the algorithms are further divided.

String matching strategies or algorithms provide key role in various real-world problems or applications. A few of its imperative applications are Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics, Digital Forensics and Information Retrieval Systems etc.

Citations:-

1. Sercan Aygün, Ece Olcay Güneş, Lida Kouhalvandi, "Python based parallel application of Knuth-Morris-Pratt algorithm", Advances in Information Electronic and Electrical Engineering (AIEEE) 2016 IEEE 4th Workshop on, pp. 1-5, 2016

2. Desi Anggreani, Desy Pratiwi Ika Putri, Anik Nur Handayani, Huzain Azis, "Knuth Morris Pratt Algorithm in Enrekang-Indonesian Language

Translator", Vocational Education and Training (ICOVET) 2020 4th International Conference on, pp. 144-148, 2020.

3. Neungsoo Park, Soeun Park, Myungho Lee, "High performance parallel KMP algorithm on a heterogeneous architecture", Cluster Computing, 2019.

3. KEYWORDS

- Parallelization
- KMP Algorithm
- Parallel Computing
- Serial and parallel parallelization

4. CONTRIBUTION OF THE WORK

The underpinning principle of the definition is that the RC component should reflect the broad range of activities and outcomes undertaken and/or achieved by a researcher relative to opportunity, and be appropriate to an individual's research discipline.

Finding reference and base papers and forwarding to the team for reading and selection. Selection of base and reference papers. Contribution divided among the three for information collection and literature survey. Worked on documentation. Comparative analysis of parallel and serial KMP algorithms, time complexity analysis of codes, implementing the code of serial KMP routine on VS Code and presentation of the same, user interfaces used, and last but not the least also worked upon the applications and future works in the given same domain.

Reading and selection of papers. Giving important ideas and searching different algorithms. Contribution in literature survey. Planned the steps of methods to be done in the future systematically, coded and executed the parallel KMP algorithm on the VS Code platform and communicated the results of the same, did research on prior algorithms, found out their significance and importance as well and even helped in the documentation.

Reading and selection of reference and base papers. Literature survey contribution. Helped in documentation, coded the serial KMP algorithm on VS Code and communicated across the results, decided the test cases, worked on the methodology of the project, did the documentation of review two, presented the same, worked upon the steps of procedures and last but not the least worked upon the parallelization of the algorithm.

All the three worked equally on the review 1,2 and 3 documents and presentation and we even talked to the guide beforehand about the topic and got the topic approved beforehand.

5. RELATED WORK- DESCRIPTION AND THE LITERATURE REVIEW TABLE

Title	Author	Journal and date	Key concepts	Advantages	Disadvantages	Future enhancement
1. Abstract Keyword D Search Ing With KMP Al Go Rtihm	Jl Jenderal Ahmad	2nd november 2017	algorithm selection is proposed for searching words	This algorithm helps in finding the words easily from the paragraph or somewhere in webpages etc.	The learning parameters used for training classifiers affects the performance of the classifiers	The user doesn't need to waste time for working on different data mining algorithms, finetuning the parameters for different algorithms.
2. KeyWorD SearChiNg WitH KMP AlgOrtiHm	Dharmapurikar et al.	9th August 2019	A Bloom filter is a space efficient probabilistic data structure that is used to test whether or not an element is a member of a set. It stores a	Knuth Morris Pratt algorithm running as expected (100% working as expected) and can serve as a tool for researchers in searching for the publication of both journals, proceedings and research reports.	The computation time involved in performing the query is independent of the number	It can be used as a tool by researchers in finding publications such as journals, proceedings or research report. The software has also been developed with a waterfall model and performed black box testing with the test results show the functional software running well as expected

			set of Signatures compactly by computing multiple hash functions on each member of the set. The answer to querying a database of strings to check for the membership of a particular string can be "false positive", but never "false negative"		of strings in the databas e, provide d the memor y used by the data structur e scales linearly with the number of strings stored in it	
3. Key wor d sear chi ng wit h KM P Alg orti hm	J. Na nd hi ni et al	17 Ja nu ary 20 17	The usage of virtualization technology can improve the utilization rate of data center equipment and bring convenience to cloud computing applications	Proposed Internet technology has developed rapidly today, the Internet has become an indispensable part of the way everyone online more and more diverse, there are PC, mobile phones, flat and even watches, etc.;	Eventhough thealgorith maccuracy,t hereisalway s a possibility that the output will fall into its wrong predictionca tegories.	the work can be extrapolated and stretched for multiclass classification.
4. Key wor d sear chi ng wit h KM P Alg orti hm	S. Vellia ngiri, P. Karthi keyan	17 De ce mb er2 01 8	Provide a systematic virus detection software solution for network security for computer systems.	This Dual port BITCAM processes next to the exact matching engine and bloom filter process. This Dual port BITCAM process is placed exclusively for obtaining higher throughput.	It will have to parse entire dataset completely many times even if the input data is the first row of training dataset	Instead of placing entire matching patterns on a chip, proposed solution is based on an antivirus processor that works as much of the filtering information as possible onto a reference memory.

5. Keyword searching with KMP Algorithm	So eun Park et.al	March2018	the performance of the KMP algorithm is limited when the input text size increases significantly beyond a certain limit.	KMP (Knuth-Morris-Pratt) algorithm is commonly used for its fast execution time compared with many other stringmatching algorithms when applied to large input texts.	However ,the performance of the KMP algorithm is limited when the input text size increases significantly beyond a certain limit.	the scope of this study has been expanded KMP algorithm to the next level.

6. Application of knuth-morris-pratt algorithm on web based document search	Amiru Fatah, Isturom Arif	June 2019	. In searching for lecture references, a document search algorithm is needed, the algorithm that can be applied to the system made is Knuth-Morris-Pratt and the development of the system using Waterfall.	Word search within the info has succeeded in knowing the position of the word that has been entered in order that the Knuth-Morris-Pratt rule will facilitate to search out student thesis documents. The system created has succeeded in displaying info regarding the quantity of scholars UN agency area unit guided by a supervisor.	The computation time involved in performing the query is independent of the number of strings in the database, provided the memory used by the data structure scales linearly with the number of	The system created will offer convenience for users in managing student thesis documents that are manually keep. The system created is correct to work out the position of the word entered by the user. Word search within the info has succeeded in knowing the position of the word that has been entered in order that the Knuth-Morris-Pratt rule will facilitate to search out student thesis documents.

					strings stored in it	
7. Optimizing the CPU-GPU memory hierarchy	Md. Raihan-Al-Masud, HossenAsiful Mustafa	December2019	The proposed parallel KMP algorithm mainly focuses on optimizing the CPU-GPU Memory hierarchy by overlapping the data Transfer between the CPU memory and the GPU memory with the string-Matching operations on the GPU.	Ensemble machine learning methods have the potential to detect and prevent different types of attacks compared to traditional machine learning methods	Drawback of not having effectiveness in measuring diversity and shows no or little relation with the accuracy of the ensemble because it only considers the difference of two models and hence, they are not valuable.	Detection rate of proposed method which is more than double compared to the existing methods.
8. Hybrid Intrusion detection system usin	Yi Tang et al	12th May 2020	The string prefix-matching algorithm follows techniques that were used in solving several	In particular, it uses the parallel string-matching algorithm of Breslauer and Galil [6] as a procedure that solves several string-matching problems	The Algorithm is not very helpful sometimes.	Reduced the error rates to 1.47% and enhanced the accuracy to 98.77% (for two-class labelling) and an average of 99.7% accuracy in labelling the four attacks.

g Machine Learning			other parallel string problems [1-3,7, 5, 7].	simultaneously and then combines the results of the string-matching problems into an answer to the string prefix matching problem.		
9. Parallel Compact Finite Automata (PC-FA)	Seunghyun Park and Jin-Young Choi	July2020	Proposed a multi-labelled hierarchical classification (MLHC) intrusion detection model that analyzes and detects external attacks caused by message injection.	we conclude that the DT and RF algorithms are applicable to high-speed internal communication environments, as well as in CAN for analysing 43 million and 46 million CAN message frames per second, respectively.	The disadvantage with this model is that, there can be loss of information and can lead to lots of classes with small number of data	the proposed MLHC model achieved high accuracy when based on the RF algorithm and rapid detection when based on the DT algorithm. Both algorithms derived. F1 scores higher than 0.998.
10.		De	This paper	Advantage is that the	Disadvanta	Deployment of the proposed

Genetic convolutional neural network for intrusion detection systems	Minh Tuan Nguyena,Kiseon Kim	cember2020	proposes an algorithm for a network intrusion detection system (NIDS) using an improved feature subset selected directly by a genetic algorithm (GA)-based exhaustive search and fuzzy C-means clustering (FCM	robust learning of the hybrid learning method containing the CNN model as an extractor and the BG classifier helped improve the final classification performance of the algorithm.	ge is that most likelihood-based methods compute the likelihood of only a few features of the data (only one), and therefore additional information that could improve accuracy of the model is ignored.	algorithm over the practical internet systems would improve the computer network security against the illegal activities.

11. Enhancing collaborative intrusion detection via disagreement-based semi-supervised learning in IoT environments	Wenjuan Li, Weizhi Meng, Man Ho Au	July2020	This paper focuses on semi-supervised learning and design DAS-CIDS, by applying disagreement-based semi-supervised (DAS) learning algorithm for Collaborative Intrusion Detection Systems (CIDSs).	The disagreement-based method could perform better than the traditional supervised learning in the aspects of both detection performance and false alarm reduction, by leveraging the unlabelled data in the training process	The disadvantage is that CIDS is vulnerable to advanced insider attacks, and the performance of semi-supervised learning can be investigated further.	supervised learning in the aspects of both detection performance and false alarm reduction, by leveraging unlabelled data in the training process
12. Effi	Xavier Larriva-	May 20	This paper presents a new model of data	Using this model required a high amount of memory in	to perform training on several	the detection of botnet attacks achieved a lower accuracy.

cient Distributed Preprocessing Model for Machine Learning-Based Anomaly Detection over Large-Scale Cybersecurity Datasets	Novo , Mario Vega-Barbas,Víctor A. Villagrá, Diego Rivera, Manuel Álvarez-Campana and Julio Berrocal	20	pre-processing based on a novel distributed computing architecture focused on large-scale datasets	order to train the pre-processed data.	GPUs at once, it is necessary to add an overhead of data to the batches that, in each epoch, are sent to each GPU to relocate the results to the complete original model.	
13. leaf-attaching	Wisam Elmasry, Akhan Akbulut, Abdul Halim Zaim	February 2017	Development of text size in the document which will be helpful for the student and faculties too to read and check the documents.	The total time taken by search pattern is going to reduces as the No. of processors increases in network. This application developed for text documents of size only MB. It may extend to any size i.e. GB to TB also and any other format likes image and video files etc.	The proposed algorithm and architecture achieve a memory efficiency of 0.56 (for the Rogets dictionary) and 1.32 (for the	The resulting set of post processed patterns can be searched using any tree search data structure. It also presents a scalable, high-throughput, Memoryefficient Architecture for large-scale String Matching (MASM) based on a pipelined binary search tree.

					Snort dictionary). As a result, our design scales well to support larger dictionaries.	
14. KMP and BMHS2	Yansen Zhou et. al.	June 2019	Improvement of time performance of pattern matching algorithm I_KMP_BMHS2	the time performance of improved pattern matching algorithm I_KMP_BMHS2 improved to some extent	The disadvantage is that running this also don't give the result near the expectation.	The paper first analyses KMP algorithm and its improved one, and then introduces BMHS2 algorithm. The distance of moving to the right of two improved algorithms is calculated when mismatch occurs respectively, and then proposes an improved algorithm based on the combination of improved KMP and BMHS2.
15. A simple fast hybrid pattern-matching algorithm	Frantisek Franek a , Christopher G. Jennings b , W.F. Smyth	16 january 2016	1. Reducing the number of letter comparisons required in the worst/average case 2. reducing the time requirement in the worst/average case	Correctness and letter comparison using the graph plotting techniques made it so easy.	For higher data it lags sometimes which is the area to be improved in the future.	Experiments indicate that in practice the new algorithm is among the fastest exact pattern-matching algorithms discovered to date, apparently dominant for alphabet size .

6. EXISTING SYSTEM DESCRIPTION

6.1 Knuth–Morris–Pratt algorithm

The KMP algorithm works on the principle that whenever a mismatchoccurs, the word itself embodies sufficient information to tell where the next match could begin thus improving the worst-case complexity. Performance: $\Theta(n)$

Advantages

a. The running time of KMP algorithm is optimal ($O(m+n)$), which is very fast compared to other string-matching algorithms.
b. The algorithm never needs to move backwards in the input text T. it makes the algorithm good for processing very large files.

Disadvantage

Doesn't work so well as the size of the alphabets increases, because more chances of mismatch occur.
Here we shall discuss the most widely used algorithms for string matching

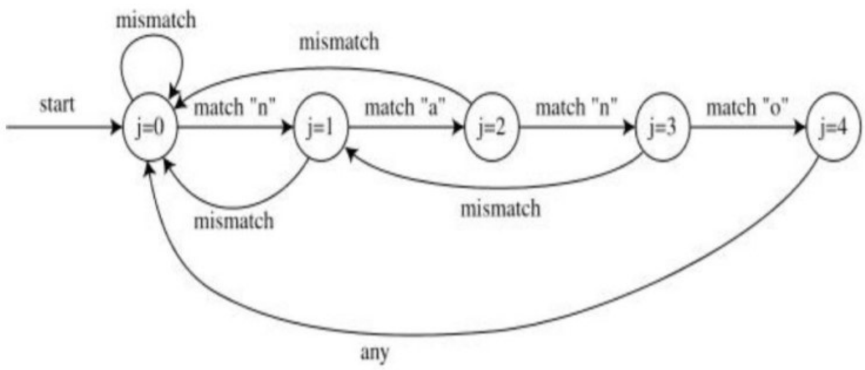

Figure 1: state transition diagram for KMP algorithm [4].

6.2 Naïve string-search algorithm

This is a very basic string-matching algorithm which maintains window that slides over the text one by one and checks for a match. If not found, the window slides one more time and checks again. This process is repeated for the rest of the document. Performance: Θ(nm)

6.3 Rabin-Karp algorithm

This algorithm uses hashing to find a set of pattern string inputted as text. It also tends to ignore case and punctuation, so it is not practical to search for a single string. A practical application of this algorithm is for plagiarism detection where many input strings in given and they can be searched for in the source material. Performance (Worst Case): Θ((n−m)m).

6.4 Boyer-Moore String-search algorithm

The Boyer-Moore Algorithm is considered the most efficient string-searching algorithm and is used as a standard benchmark for comparing the efficiency of other algorithms. This algorithm pre-processes the string being searched for and uses information gathered during the pre-processing step to skip sections of text and has better performance as the input string gets longer.

Algorithm	Preprocessing time	Matching time[1]	Space
Naïve string-search algorithm	none	Θ(nm)	none
Rabin–Karp algorithm	Θ(m)	average Θ(n + m), worst Θ((n−m)m)	O(1)
Knuth–Morris–Pratt algorithm	Θ(m)	Θ(n)	Θ(m)
Boyer–Moore string-search algorithm	Θ(m + k)	best Ω(n/m), worst O(mn)	Θ(k)
Bitap algorithm (*shift-or, shift-and, Baeza–Yates–Gonnet*)	Θ(m + k)	O(mn)	
Two-way string-matching algorithm	Θ(m)	O(n+m)	O(1)
BNDM (Backward Non-Deterministic Dawg Matching)	O(m)	O(n)	
BOM (Backward Oracle Matching)	O(m)	O(n)	

Table 1: comparative analysis of various existing word search algorithms

7. PROPSED SYSTEM ARCHITECTURE AND MODULEWISE DESCRIPTION

7.1 METHODOLOGY

Here we are going to discuss about 'Knuth–Morris–Pratt-algorithm' and how it helps solving the problem of word or string searching in an algorithm. In both serial and parallel computing, this algorithm can be implemented, and the differences can be noticed on both ends. Thus, the conclusion of this project will be implementing the KMP algorithm in both serial and parallel way and thus concluding which is faster.

Methods to be followed for implementation:

1. Installing Python on Windows
2. Parallelization in C++
3. Parallel Execution
4. .Sequential Execution
5. Image Searching using KMP
6. Evaluation Result

It would be an entire practical approach of implementation.

7.2 Tools description and execution

1. Download the Dev C++ installer from a trusted source such as the official Bloodshed website or SourceForge.
2. Run the installer and choose the language you want to use for the installation process.
3. Read and accept the license agreement and then choose the components you want to install. By default, all components are selected, but you can deselect any that you do not need.
4. Choose the installation folder for Dev C++. By default, it will be installed in the C:\Dev-Cpp folder, but you can change it if you prefer.
5. Choose whether you want to create a desktop shortcut or not.
6. Choose the additional tasks you want to perform. By default, "Create associations" and "Add to PATH" are selected. If you are not sure what these options mean, it is safe to leave them selected.
7. Click "Install" and wait for the installation process to complete.
8. Once the installation is complete, click "Finish" to exit the installer.
9. Launch Dev C++ by double-clicking on the desktop shortcut or by going to the installation folder and opening the DevCpp.exe file.
10. When you launch Dev C++ for the first time, you will be prompted to configure the compiler settings. Follow the prompts to set up the compiler according to your preferences.

8. CODE

8.1 sample text for pattern matching

https://www.gutenberg.org/files/1661/old/advsh12h.htm

8.2 simple serial KMP algorithm

```
#include <ctime>
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <sstream>

using namespace std;

void preKMP(char *pattern, int failure[]);
void KMP(char *target, char *pattern, int *failure, int *answer, int pattern_length, int target_length);

int main(int argc, char **argv)
{
    fstream target_file("target.txt"), pattern_file("pattern.txt");
    stringstream target_stream, pattern_stream;

    target_stream << target_file.rdbuf();
    string target_string = target_stream.str();

    pattern_stream << pattern_file.rdbuf();
    string pattern_string = pattern_stream.str();
    int target_length = target_string.length();
    int pattern_length = pattern_string.length();

    char *target = new char[target_length + 1];
    char *pattern = new char[pattern_length + 1];

    strcpy(target, target_string.c_str());
    strcpy(pattern, pattern_string.c_str());

    int *failure = new int[pattern_length];
    int *answer = new int[target_length]();
```

```
    preKMP(pattern, failure);

    struct timespec start, end;
    double elapsed_time;

    cout << "----- This is sequential results using KMP Algorithm. -----" << endl;

    clock_gettime(CLOCK_MONOTONIC, &start);
    KMP(target, pattern, failure, answer, pattern_length, target_length);
    clock_gettime(CLOCK_MONOTONIC, &end);

    elapsed_time = (end.tv_sec - start.tv_sec) * 1e3 + (end.tv_nsec - start.tv_nsec) / 1e6;
    cout << "When the target length is " << target_length << ", pattern length is " << pattern_length << ", the elapsed
time is " << elapsed_time << " ms." << endl;

    int counter = 0;

    for (int i = 0; i < target_length; i++)
    {
        if (answer[i])
        {
            cout << "Find a matching substring starting at: " << i << "." << endl;
            counter++;
        }
    }

    cout << counter << endl;

    delete[] target;
    delete[] pattern;
    delete[] failure;
    delete[] answer;

    return 0;
}

void KMP(char *target, char *pattern, int *failure, int *answer, int pattern_length, int target_length)
{
    int i = 0, j = target_length;

    int k = 0;
    while (i < j)
    {
        if (k == -1)
        {
            i++;
            k = 0;
```

```
        }
        else if (target[i] == pattern[k])
        {
            i++;
            k++;

            if (k == pattern_length)
            {
                k--;
                answer[i - pattern_length] = 1;
                i = i - pattern_length + 1;
            }
        }
        else
            k = failure[k];
    }

    return;
}

void preKMP(char *pattern, int failure[])
{
    int m = strlen(pattern);
    int k;

    failure[0] = -1;

    for (int i = 1; i < m; i++)
    {
        k = failure[i - 1];

        while (k >= 0)
        {
            if (pattern[k] == pattern[i - 1])
                break;
            else
                k = failure[k];
        }

        failure[i] = k + 1;
    }

    return;
}
```

8.3 Parallel KMP algorithm execution

```
#include <ctime>
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <sstream>
#include <pthread.h>

#define CORE_NUM 4

using namespace std;

pthread_mutex_t lock;
class sending_parameter
{
public:
   char *target;
   char *pattern;
   int *failure;
   int *answer;
   int pattern_length;
   int target_length;
   int part_start;
   sending_parameter(char *i, char* j, int* k, int m, int n, int o) :
   target(i), pattern(j), failure(k), pattern_length(m), target_length(n), part_start(o) {}
} ;


void preKMP(char *pattern, int failure[]);
void *KMP(void* info);
int *answer;

int main(int argc, char **argv)
{
   fstream target_file("target.txt"), pattern_file("pattern.txt");
   stringstream target_stream, pattern_stream;

   target_stream << target_file.rdbuf();
   string target_string = target_stream.str();

   pattern_stream << pattern_file.rdbuf();
   string pattern_string = pattern_stream.str();

   int target_length = target_string.length();
   int pattern_length = pattern_string.length();

   char *target = new char[target_length + 4];
   char *pattern = new char[pattern_length + 4];

   strcpy(target, target_string.c_str());
   strcpy(pattern, pattern_string.c_str());

   int *failure = new int[pattern_length];
   answer = new int[target_length]();

   preKMP(pattern, failure);

   struct timespec start, end;
   double elapsed_time;
```

```
    cout << "----- This is pthread results using KMP Algorithm. -----" << endl;

    clock_gettime(CLOCK_MONOTONIC, &start);

    pthread_t* threads;
    threads = (pthread_t*)malloc(CORE_NUM * sizeof(pthread_t));

    int first_partial = target_length / CORE_NUM;
    // first round
    for (int i = 0; i < CORE_NUM; i++)
    {
      int part_start = i * first_partial;
      auto info_struct = new sending_parameter(target, pattern, failure, pattern_length, first_partial, part_start);
      pthread_create(&threads[i], NULL, KMP, (void*)info_struct);
    }

    for (int thread = 0; thread < CORE_NUM; thread++){
            pthread_join(threads[thread], NULL);
    }


    // second round
    int second_partial = (pattern_length - 1) * 2;
    for (int i = 0; i < CORE_NUM - 1; i++)
    {
      int part_start = (i + 1) * first_partial - (pattern_length - 1);
      auto info_struct = new sending_parameter(target, pattern, failure, pattern_length, second_partial, part_start);
      pthread_create(&threads[i], NULL, KMP, (void*)info_struct);
    }

    for (int thread = 0; thread < CORE_NUM - 1; thread++){
            pthread_join(threads[thread], NULL);
    }

    // last part
    int last_partial = (target_length % CORE_NUM) + pattern_length - 1;
    if(last_partial != 0)
    {
      int part_start = CORE_NUM * first_partial - (pattern_length - 1);
      auto info_struct = new sending_parameter(target, pattern, failure, pattern_length, last_partial, part_start);
      KMP((void*)info_struct);
    }
    clock_gettime(CLOCK_MONOTONIC, &end);

    elapsed_time = (end.tv_sec - start.tv_sec) * 1e3 + (end.tv_nsec - start.tv_nsec) / 1e6;
    cout << "When the target length is " << target_length << ", pattern length is " << pattern_length << ", the elapsed
  time is " << elapsed_time << " ms." << endl;

    int counter = 0;

    for (int i = 0; i < target_length; i++)
    {
      if (answer[i])
      {
        cout << "Find a matching substring starting at: " << i << "." << endl;
        counter++;
      }
    }

    cout << counter << endl;
```

```
    delete[] target;
    delete[] pattern;
    delete[] failure;
    delete[] answer;

    return 0;
}


void *KMP(void* info)
{
    sending_parameter info_struct = *(sending_parameter*) info;

    char *target = info_struct.target;
    char *pattern = info_struct.pattern;
    int *failure = info_struct.failure;
    int part_start = info_struct.part_start;
    int target_length = info_struct.target_length;
    int pattern_length = info_struct.pattern_length;

    int i = part_start, j = part_start + target_length;

    int k = 0;

    while (i < j)
    {
        if (k == -1)
        {
            i++;
            k = 0;
        }
        else if (target[i] == pattern[k])
        {
            i++;
            k++;

            if (k == pattern_length)
            {
                k--;
                answer[i - pattern_length] = 1;
                i = i - pattern_length + 1;
            }
        }
        else
            k = failure[k];
    }

    return NULL;
}

void preKMP(char *pattern, int failure[])
{
    int m = strlen(pattern);
    int k;

    failure[0] = -1;

    for (int i = 1; i < m; i++)
    {
        k = failure[i - 1];

        while (k >= 0)
```

```
        {
            if (pattern[k] == pattern[i - 1])
                break;
            else
                k = failure[k];
        }

        failure[i] = k + 1;
    }

    return;
}
```

9. RESULTS

S.No	Pattern word searched for	Serial KMP (ms)	Parallel KMP (ms)
1.	sher	25.0329	10.7226
3.	man	33.5945	13.5154
4.	pattern	18.292	7.2538
5.	hammer	25.3718	11.8196
6.	concern	19.1464	14.0371
7.	arrest	63.5469	12.7825
8.	sherlock	27.4855	11.0079
9.	premises	19.3357	11.2569

From the result we got, we can clearly say that KMP algorithm used in parallel computing gives faster result than compared to its implementation in serial.

10. CONCLUSION

Through the experimental demonstration of string-matching processing, it is not uneasy to discover that the efficiency of serial KMP and parallel KMP algorithms are almost equal, assuming that the amount of data is small. However, when the data set is large, and the number of types of the pattern is relatively big, or the number is small but unevenly distributed, the parallel KMP algorithm is superior to the serial KMP algorithm. This study confirms that string matching problems can be solved in practical applications. Additionally, the new algorithm can be improved in many aspects by optimizing the last-identical array or completing the comparing process with the letter numbered table.

Applications that can be developed and improved using these algorithms include: A few of its imperative applications are Spell Checkers, Spam Filters, Intrusion Detection System, Search Engines, Plagiarism Detection, Bioinformatics, Digital Forensics, and Information Retrieval Systems etc.

11. REFERENCES

1. A. Apostolico and D. Breslauer, an optimal O(loglogn) time parallel algorithm for detecting all squares in a string, Tech. Report CUCS-040-92, Computer Science Dept., Columbia Univ., 1992.

2. A. Apostolico, D. Breslauer and 2. Galil, Optimal parallel algorithms for periods, palindromes, and squares, in: Proc. 19th Internat. Coil. on Automata, Languages, and Programming, Lecture Notes in Computer Science Vol. 623 (Springer, Berlin, 1992) 296307.

3. A. Apostolico, D. Breslauer and Z. Galil, Parallel detection of all palindromes in a string, Theorer. Comput. Sci., to appear.

4. D. Breslauer, Efficient string algorithmics, Ph.D. Thesis, Dept. of Computer Science, Columbia Univ., New York, NY, 1992.

5. D. Breslauer, Testing string super primitivity in parallel, Inform. Process. Lett. 49 (1994) 235-241.

6. D. Breslauer and Z. Galil, An optimal O (log log n) time parallel string- matching algorithm, SIAM J. Comput. 19 (1990) 1051-1058.

7. D. Breslauer and Z. Galil, finding all periods and initial palindromes of a string in parallel, Algorithmica, to appear.

8. N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic Memory- Efficient String-Matching Algorithms for Intrusion Detection. In Proceedings of IEEE Infocom, Hong Kong, March 2004.

9. S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. IEEE Micro, 24(1):52–61, 2004.

10. J.Nandhini, Dr.M. Nithya, Dr.S.Prabhakaran "Advance virus detection using combined techniques of pattern matching and dynamic instruction sequences", International Journal of Communication and Computer Technologies, Volume 01 – No.45, pp.156-161 2013.

11. HyunJin Kim et al.," A Memory-Efficient Bit-Split Parallel String-Matching using Pattern Dividing for Intrusion Detection Systems", IEEE Transactions On Parallel And Distributed Systems, Third Draft, September 2010, pp.1-8,2011.

12. Yi Tang et al.," Independent Parallel Compact Finite Automatons for Accelerating Multi-String Matching", IEEE Globecom 2010 proceedings, 2010.

13. KSMV Kumar, S. Viswanadha Raju and A. Govardhan, "Overlapped Text Partition Algorithm for Pattern Matching on Hypercube Networked Model", GJCST, pp. 1-8,2013.

14. Hoang Le, and Viktor K. Prasanna, "A Memory Efficient and Modular Approach for Large-Scale String Pattern Matching", IEEE Transactions on Computers, VOL. 62, NO. 5, pp. 844-857, 2013.

15. Y. Zhou and R. Pang, "Research of Pattern Matching Algorithm Based on KMP and BMHS2," *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2019, pp. 193-197, doi: 10.1109/ICCC47050.2019.9064076.

16. S. Park, D. Kim, N. Park and M. Lee, "High Performance Parallel KMP Algorithm on a Heterogeneous Architecture," *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Trento, 2018, pp. 65-71, doi: 10.1109/FAS-W.2018.00027.

17. Q. Meng, Z. Lei, D. He and H. Wang, "Application of KMP algorithm in customized flow analysis," *2017 3rd IEEE International Conference on Computer* *and Communications (ICCC)*, Chengdu, 2017, pp. 2338-2342, doi: 10.1109/Comp

18. Websites including google.com, wikipedia.org and image searches in the google search engine for the searches of spell checker, plagiarism detector and intrusion detection interfaces.